Thank you to Andrew Fitzgerald, Harsh Patel, Jon Charbonneau, Kevin Galler, Lanre Ige, Mert Mumtaz, Pranav Garimidi, Ryan Chern, Tao Zhu, and Tarun Chitra for feedback and review.

Introduction

Eclipse is Ethereum's first SVM L2. We're incredibly excited to bring the existing SVM's power to more users, but we're also dedicated to pushing forward ongoing R&D around the SVM itself. We're focused on ensuring that Eclipse's development undeniably returns value back to all SVM chains, especially Solana.

As a precursor to future articles on our thinking around fee markets, this article will analyze Solana's existing fee market and associated proposals to improve it. We frame these proposals alongside the primary theoretical goals for any Transaction Fee Mechanism (TFM), borrowing heavily from the work of Tim Roughgarden, Maryam Bahrani, Pranav Garimidi, Hao Chung, Elaine Shi, and others. We'll indicate core definitions throughout with **

.

In general, a TFM determines:

1. Which transactions are included in a given block,
2. The fees a given transaction pays, and
3. How (and to whom) the accumulated transaction fees are distributed.

Which transactions are included in a given block,

The fees a given transaction pays, and

How (and to whom) the accumulated transaction fees are distributed.

Ultimately, this article aims to combine the wealth of Ethereum-centric TFM research with Solana's innovative engineering.

Overview of Solana & Ethereum Current TFMs

Solana vs. Ethereum Basics

We'll begin with an overview of Solana's TFM and contrast it to Ethereum's. This will better contextualize the relevant proposals so that we can work towards modifying and improving the TFM. For starters:

Solana's base fee is a fixed 5,000 lamports (0.000005 SOL) per signature,

and most transactions have one signature. It does not account for a transaction's broader computational resources (as measured by CUs).

Solana Tx Base Fee = (5,000 Lamports) x (# of Signatures in Tx)

Ethereum's base fee mechanism differs in two main ways:

- Dynamic
- Ethereum's base fee (as measured by gwei per unit of gas) floats based on trailing market demand.
- More granular charge per unit of computation
- Ethereum's per transaction base fee is linear in the amount of gas consumed.

Dynamic

- Ethereum's base fee (as measured by gwei per unit of gas) floats based on trailing market demand.

More granular charge per unit of computation

- Ethereum's per transaction base fee is linear in the amount of gas consumed.

So Ethereum's per transaction base fee is:

Ethereum Tx Base Fee = (Prevailing gas price in gwei) x (gas used in tx)

Solana users can also add optional priority fees to improve their inclusion probability. Unlike the base fee, the priority fee is priced per CU requested

for a transaction. Solana transactions can include the following Compute Budget instructions:

- SetComputeUnitLimit
- Transactions can set a maximum amount of CUs that they are allowed to consume (with a max of 1.4M CUs per transaction). When executed, the transaction can use up to that requested CU limit. If no SetComputeUnitLimit

instruction is provided, the transaction's CU limit is calculated as (# of instructions in the transaction) x 200k CUs).

- SetComputeUnitPrice
- **of micro-lamports per CU requested that the transaction offers to pay in its priority fee.**

SetComputeUnitLimit

- Transactions can set a maximum amount of CUs that they are allowed to consume (with a max of 1.4M CUs per transaction). When executed, the transaction can use up to that requested CU limit. If no SetComputeUnitLimit

instruction is provided, the transaction's CU limit is calculated as (# of instructions in the transaction) x 200k CUs).

SetComputeUnitPrice

- **of micro-lamports per CU requested that the transaction offers to pay in its priority fee.**

Putting them together:

Tx Priority Fee = (Tx CU Limit) x (CU Price)

Note that this priority fee is paid against the full CUs requested (regardless of whether the transaction uses the total amount requested), unlike in Ethereum, where the fee is a function of how much gas the transaction actually uses.

Fee Burn vs. Validator Rewards

While the incentive for validators to prioritize high-fee transactions sits outside of consensus

, it's enforced in consensus

that both the base fee and priority fee are 50/50 burned/sent to the leader (current block producer) in Solana:

- Base Fee

— Mandatory for block inclusion. Transactions lacking the requisite base fee will be rejected.

- Priority Fee

— Not mandatory for block inclusion. Used to optionally prioritize transactions that want to increase their probability of fast inclusion.

Base Fee

— Mandatory for block inclusion. Transactions lacking the requisite base fee will be rejected.

Priority Fee

— Not mandatory for block inclusion. Used to optionally prioritize transactions that want to increase their probability of fast inclusion.

A user cannot avoid paying the base fee, but one can avoid the priority fee and signal their desire to be prioritized in another manner. We already see this in practice — Jito-Solana auctions pay 100% (less a fee) to the leader out-of-band. SIMD-0096 offers a simple fix to this issue, rewarding 100% of the priority fee to the validator.

Direct Transfer

Importantly, Solana validators cast votes for each block with on-chain transactions. They pay the base fee for each of these transactions.

Solana has been generating its all-time highs for network fees lately, driven by a sharp increase in priority fees. Recent fee splits are shown below:

Source: Solana Compass

Ethereum Block Builder vs. Solana Scheduler

Ethereum's block production is generally easier to understand, so we'll start there. Nearly all validators (a.k.a. proposers) outsource block production to out-of-protocol builders via MEV-Boost. Builders create blocks every 12 seconds (Ethereum's slot time) and pass these whole blocks to the proposer (via a relay), who selects the highest-value block.

In both Ethereum and Solana, block producers have the power to order transactions arbitrarily within a block. They are incentivized to do so in a manner that maximizes their profit. For example, different Ethereum builders can compete by running proprietary algorithms that more effectively maximize profits versus competitors.

This means that even in Ethereum, sending a high-priority fee does not

achieve any in-protocol deterministic guarantee of block inclusion

or ordering

. However, it is very likely to achieve the expected result due to the nature of Ethereum's current block-building process in which the builder builds a full profit-maximizing block at the end of each discrete slot.

For example, a searcher may send an arbitrage transaction with an incredibly high priority fee (e.g., higher than all other eligible transactions combined) to the builder, requesting its inclusion at the top of the block and to exclude the transaction from the block entirely if it doesn't get the top of block position. In this case, a rational profit-maximizing builder will include this transaction at the top of the block even if they only received it right towards the end of their 12-second slot.

You'll note that there are two different guarantees that fees are trying to achieve here:

- Inclusion
- A user wants to get their transaction included in this block, but they don't care where it lands in the block.
- Ordering
- A user doesn't just want to be included in the block anywhere; they want priority access to a specific state at a given time.

Inclusion

- A user wants to get their transaction included in this block, but they don't care where it lands in the block.

Ordering

- A user doesn't just want to be included in the block anywhere; they want priority access to a specific state at a given time.

Ethereum's EIP-1559 mechanism has proven quite effective at enabling users to easily bid for block inclusion with a high probability of success. There's a global reserve price that everyone knows to pay, and paying it (usually alongside a nominal priority fee) should reliably get a user's transaction included promptly. However, the mechanism does not seek to provide any assurances around ordering (i.e., priority access to state), whereas out-of-protocol mechanisms are reliable for users seeking such assurances (directly from builders, for example).

Solana's block-building process works very differently. Validators do not outsource full block production in discrete time slots to out-of-protocol builders. The "scheduler" is the default algorithm included in the Solana Labs validator client, which schedules transactions for execution and builds blocks continuously.

Additionally, Solana transactions specify what accounts must be read- and write-locked for execution. This allows the scheduler to iteratively sort which transactions can be executed concurrently — as transactions that don't touch the same state can be executed in parallel.

Within a block, a maximum of 12,000,000 CUs can be used for sequential writes to a single account ("piece of state"). This is roughly the amount of CUs that can be processed by a single thread per 400ms slot on reasonable node requirements. Solana's per-block limit is then 48,000,000 CUs. The current scheduler implementation uses four threads for non-vote transactions, and 12M x 4 = 48M. In theory, this means using more cores = increasing CU limits. Scaling with hardware.

The scheduler non-deterministically prioritizes transactions with higher priority fees. However, this generally provides less reliable prioritization guarantees than mechanisms such as those described in the case of Ethereum today.

In Solana, validators running the default scheduler build blocks continuously

, so transactions can be added to an in-progress block and executed before waiting until the slot's end to organize them solely by priority fee. The intention is for the scheduler to very roughly

profit-maximize by prioritizing transactions based on their total CU price.

Solana's default multi-threaded scheduler also introduces additional "jitter." Transactions are assigned randomly to one of the four threads, with each thread maintaining its own queue of transactions waiting for execution. Priority fees are then used to prioritize transactions intra-thread. However, they do not help prioritize transactions inter-thread.

For example, two searchers may each send a transaction simultaneously to capture the same arbitrage opportunity, and the one that sends a lower

priority fee may even win because they land in a less clogged queue by chance. This reduces the efficacy of priority fees and increases the incentive to spam relative to Ethereum — especially since inclusion for transactions is also dependent on when, within a given slot, a transaction gets to the current block producer.

Note that there are planned changes to Solana's default scheduler, which aim to address some of the issues with the current implementation by relying on a graph of transaction dependencies and scheduling the highest priority unblocked (non-write-locked) transactions in the graph — we cover this later in the article.

While mostly outside this article's scope, the Jito-Solana client allows searchers to capture miner/maximal extractable value (MEV) more efficiently in ways that minimize negative externalities on Solana. Jito-Solana deviates from Solana's default scheduler by introducing out-of-protocol discrete 200-millisecond Flashbots-esque bundle auctions, which are run in parallel to the default continuous block production and a private mempool (which again deviates from Solana's default TFM). The adoption of the Jito-Solana client by Solana validators (>50% of validators run it today) has helped tackle some of the issues with Solana's existing TFM — namely, the prevalence of MEV-driven spam.

Shortcomings of Solana's Current TFM

While Solana's TFM is highly promising, it also has some potential shortcomings for now:

Incentive to Spam

As mentioned above, transactions are ordered in somewhat of a first in, first out (FIFO) manner as soon as they reach the block producer. Additionally, they're subject to non-determinism from both network jitter and the randomized thread allocation of the default scheduler. Although priority fees may help inclusion probability in certain circumstances, a substantial incentive still exists to spam transactions to maximize the fastest inclusion probability (e.g., a searcher racing to liquidate an in-default position on a lending market). The image below from Jito Labs helps demonstrate the suboptimal nature of spam transactions.

Source: Jito Foundation

First-Price Auction

In a naive first-price auction (FPA), users just submit bids, with the highest getting included in the block. One issue with FPAs is that they're not very user-friendly. Users must guess what other users are bidding, thinking about what they're willing to bid up to, potentially shading their bid lower so as not to overpay based on what they believe others are bidding, for example.

More formally, the FPA model is non-DSIC:

**Dominant Strategy Incentive Compatible (DSIC)

: Assuming that the block producer implements the TFM honestly, then the prescribed bidding strategy should be the dominant strategy for users. This means users will bid (in transaction fees) at the exact value they ascribe to transaction inclusion [Chu22].

DSIC is one of the key properties that the creators of EIP-1559 aimed to introduce into Ethereum's TFM with its implementation, and as we described earlier, it can be considered a success. Users more easily know the public reserve price to be included in the block at a given time (via the dynamic base fee), so paying it (plus any nominal priority fee) will almost always get your transaction included quickly.

Conversely, Solana's TFM is a naive FPA. It lacks a reliable mechanism for users to accurately express their preference for block inclusion and is non-DSIC. In practice, trying to set exactly the right priority fee at the right time is incredibly challenging. This disproportionately favors sophisticated participants who are more capable of bypassing network and scheduler jitter (e.g., via co-location or spamming transactions).

50/50 Burn/Validator Payout

As noted earlier, Ethereum burns 100% of the base fee while sending 100% of the priority fee to the block producer, while for Solana, both the base fee and priority fee are 50/50 burned/paid to the block producer. As a result of this, the Solana TFM is non-OCA-proof:

**Off-Chain Agreement-proofness (OCA-proofness or SCP)

: No off-chain agreement between users and the block producer can Pareto improve the outcome of the TFM for a given block [Rou21]. A c-SCP protocol is resistant to a coalition of the block producer and up to c

users being able to profit by deviating from reporting truthfully.

We see a clear example of this with Jito-Solana's out-of-protocol auctions paying out 100% of bids (less Jito's cut) to the block producers, rather than burning 50% — Jito-Solana is an example of an Off-Chain Agreement used by block producers. However, we note that Jito-Solana tips are not equivalent to priority fees, as the former are only paid if the associated transaction (and bundle) successfully executes.

The recently proposed SIMD-0109 would introduce a tipping mechanism (similar to that used by Jito-Solana's out-protocol-auction) into the protocol as a native instruction.

Lack of Privileged Transaction Types

Solana vote transactions are posted on-chain and must be included in blocks, yet each validator must pay for the costs of said transactions. This represents a significant fixed cost (paid for privately by validators) despite the positive externality of the inclusion of vote transactions. This cost is exacerbated by the fact that vote transactions are overcharged relative to the CUs consumed (i.e., they use relatively few CUs vs. the average transaction). The economics create a centralizing effect here, as the total voting cost is roughly constant for any validator while the rewards earned are proportional to stake weight.

Source: Ceteris, Solana the Monolith

As an aside, similar logic could be extended to include reliable oracle updates, which networks typically charge for despite the positive externality of accurate on-chain price feeds. A more opinionated chain that derives high value from a particular robust oracle may choose to enshrine a mechanism that subsidizes its cost.

Solana's Local Fee Market

Solana's approximation of a local fee mechanism exists because no account can write more than 12M CUs per 48M block limit. This, alongside the multi-threaded nature of Solana's default scheduler, means that a maximum of 25% of transactions in a block can correspond to a single piece of in-demand state. In theory, users of less-in-demand state should not have to increase their priority fees for strong inclusion guarantees compared to users of in-demand state.

This is arguably not a genuine local fee mechanism. The mechanism is not enforced by consensus (it's only at the scheduler level), and the relationship between priority fees and block inclusion is rather non-deterministic (as previously mentioned). It also lacks a notion of 'elasticity' where both target and maximum resource limits exist.

Inefficient CU Usage & Requests

Because Solana's base fee does not account for CUs, it does not incentivize transactions to:

- Use CUs efficiently

— A transaction with 1.4M CUs carries the same base fee as one with 100k CUs, all else equal.

- Request CUs efficiently

— Even if a transaction uses 50k CUs, it carries the same base fee whether it requests 100k CUs or 1M CUs.

Use CUs efficiently

— A transaction with 1.4M CUs carries the same base fee as one with 100k CUs, all else equal.

Request CUs efficiently

— Even if a transaction uses 50k CUs, it carries the same base fee whether it requests 100k CUs or 1M CUs.

This can lead to an overestimation of compute needed within a given block by the scheduler and an efficiency loss compared to resources required by the block producer at a given slot. A DSIC TFM would fix this issue as a user's dominant strategy would be that of the prescribed bidding strategy — in this case, accurately representing the expected usage of CUs.

Costless to Write Lock Accounts

As mentioned earlier, Solana transactions specify upfront all accounts they will read or write to while executing. However,

this mechanism can be abused today to globally lock any account at effectively no cost. For example:

- I send T

x

A

Tx_{A}

T

x

A

which specifies that it will write to A

c

c

o

u

n

t

A

Account_{A}

A

cco

u

n

t

A

.

- The leader receives T

x

A

Tx_{A}

T

x

A

, schedules it, and starts to execute it. A

c

c

o

u

n

t

A

Account_{A}

A

cco

u

n

t

A

is now locked - no other transaction touching A

c

c

o

u

n

t

A

Account_{A}

A

cco

u

n

t

A

can be executed until T

x

A

Tx_{A}

T

x

A

has finished executing.

I send T

x

A

Tx_{A}

T

x

A

which specifies that it will write to A

c

c

o

u

n

t

A

Account_{A}

A

cco

u

n

t

A

.

The leader receives T

x

A

Tx_{A}

T

x

A

, schedules it, and starts to execute it. A

c

c

o

u

n

t

A

Account_{A}

A

cco

u

n

t

A

is now locked - no other transaction touching A

c

c

o

u

n

t

A

Account_{A}

A

cco

u

n

t

A

can be executed until T

x

A

Tx_{A}

T

x

A

has finished executing.

The problem stems from the fact that anyone can send transactions that write-lock any accounts they want. There's no cost to locking accounts, and transactions can even lock accounts that they don't use, which is a clear spam attack vector. Moreover, account owners don't have control over who can write-lock their own account.

TFM Proposals & Frameworks

Every blockchain must ultimately decide how to allocate the scarce resource of its finite blockspace among users, which it does via its TFM. Below, we will discuss several relevant TFM proposals and frameworks that may be valuable for Solana.

Multidimensional Blockchain Fee Markets

Most existing fee markets are one-dimensional, built around a single fungible unit of account (e.g., gas in Ethereum). However, this single resource purchased is a proxy for many underlying non-fungible resources (e.g., bandwidth, computation, and storage).

For example, each Ethereum opcode carries a certain fixed amount of gas it consumes (e.g., ADD uses 3 gas, while MUL uses 5). The gas price for each opcode was set as a rough approximation of the underlying resources they use and how expensive those are deemed to be for nodes in the network. For example, this implicit measure of an operation's cost can be determined by running benchmarks on real-world hardware.

However, it's also possible to construct multidimensional fee markets that individually price these different non-fungible resources rather than combining them into one unit. EIP-4844 is a straightforward two-dimensional fee market, as data blobs have their own fee market independent of Ethereum execution gas.

This 2022 paper by Diamandis, Evans, Chitra, and Angeris analyzes how to construct multidimensional fee markets such as this. Their work frames the TFM construction problem from the perspective of a network designer aiming to maximize the welfare (or total utility) of the users of the blockchain minus the resource consumption of said users, subject to the chain's transaction and block constraints (for example, smart contract limits or CU/gas limits). The main result of the paper is that despite the welfare being unknown, they design a mechanism that maximizes it and show how to explicitly construct said mechanism.

**Welfare Maximizing

: The intended allocation and payment rules imply that the sum of consumer and miner surplus is (approximately) maximized.

Their key finding is that an equivalent TFM is implementable, which is one where the allocation rule is set such that the participants in the mechanism are incentized to act in accordance with the auction's intended behavior. This work demonstrates that pricing resources separately can make a blockchain more efficient and more resilient to periods of high congestion or spam. It is implemented in practice for both Avalanche and Penumbra. Controller-based base fee mechanisms, such as EIP-1559, are highlighted as a potential approach that could work exceptionally well on Solana and SVM chains, given the short block times, allowing the base fee to adjust quickly to changes in user demand and resource availability.

As previously mentioned, one conclusion from the paper is that it is possible to design systematic and computationally efficient ways to help define and update the pricing of multidimensional resources for blockchains. However, a natural question should be: what resources would make sense to price distinctly? Some practical work has been done within other blockchain contexts to make such decisions. For example, Penumbra has implemented a form of multidimensional resource pricing to price resources used by full nodes and end-user devices separately on their privacy-centric blockchain.

While the 2022 paper generally discusses multidimensional pricing of base resources (e.g., compute, bandwidth, storage), it is also possible to implement multidimensional resource pricing per account (i.e., per "piece of state"). Each account is treated as a different resource. This is discussed in this recent article, which builds on the original paper. Individually pricing accounts (rather than compute, storage, bandwidth, etc.) as the underlying resource may also be more straightforward to implement with a reduced risk of resource exhaustion attacks.

Exponential Fee for Write Lock Accounts

Following Anatoly's recent post on SVM Execution Economics, Tao Zhu, in collaboration with Anatoly, proposed SIMD-0110. Its primary motivation is to deter spam with economic back pressure (i.e., increasing fees in a targeted manner over time to reduce the incentive to spam), resulting in more efficient network resource utilization. Failed arbitrage transactions continue to fill up roughly half (or more) of Solana's blockspace because it's rational and incredibly cheap to spam.

The proposal recommends tracking the Exponential Moving Average (EMA) of each account's CU utilization per block to achieve this. The cost to write-lock an account would increase exponentially based on its respective trailing CU utilization, deterring spam. The core logic is similar to how EIP-1559 sets Ethereum's global base fee as a function of gas usage in trailing blocks. However, this SIMD is far more granular in setting per-account local base fee markets.

The basic implementation idea for the account-based varying write-lock fee would be as follows:

- Track each contentious account's trailing EMA compute-unit utilization

over the past 150 slots.

- The maximum number of accounts that will be tracked is 2048, where only the most contentious accounts with the highest write-lock cost rate

are tracked.

- If an account's EMA compute-unit utilization

is >50% of its max CU limit, its write-lock cost rate

will increase by X%. If it's <50% of its limit, the cost rate

will decrease by X%.

- V0 recommends an initial write-lock cost rate of 1000 micro-lamports/CU and a cost rate

adjustment rate of 1% per slot (note that the exact percentages here are all subject to change given the early nature of the proposal).

- The write-lock fee for an account at a given block is calculated by multiplying the write-lock cost rate by the transaction's requested CUs.
- Transactions continue to pay signature fees, and optional priority fees also remain.
- Collected write-lock fees are 100% burnt.
- Collected priority fees are 100% rewarded.
- Collected signature fees are 50% burnt and 50% rewarded.

Track each contentious account's trailing EMA compute-unit utilization

over the past 150 slots.

The maximum number of accounts that will be tracked is 2048, where only the most contentious accounts with the highest write-lock cost rate

are tracked.

If an account's EMA compute-unit utilization

is >50% of its max CU limit, its write-lock cost rate

will increase by X%. If it's <50% of its limit, the cost rate

will decrease by X%.

V0 recommends an initial write-lock cost rate of 1000 micro-lamports/CU and a cost rate

adjustment rate of 1% per slot (note that the exact percentages here are all subject to change given the early nature of the proposal).

The write-lock fee for an account at a given block is calculated by multiplying the write-lock cost rate by the transaction's requested CUs.

Transactions continue to pay signature fees, and optional priority fees also remain.

Collected write-lock fees are 100% burnt.

Collected priority fees are 100% rewarded.

Collected signature fees are 50% burnt and 50% rewarded.

This proposal would make the write-lock feature of Solana (usually) DSIC similar to how EIP-1559 made Ethereum's TFM (usually) DSIC and MMIC [Rou23] — except in the presence of sudden fee spikes.

We can define the MMIC property as follows:

**Myopic Miner Incentive Compatibility (MMIC)

: The block producer maximizes its utility by creating no fake transactions and following the stated rules of the TFM. Myopic means this goal only concerns the current block when judging utility maximization [Rou21].

Any trailing mechanism is imperfect in that it may not accurately represent the exact current state of demand. For example, demand may be low for an extended period of time (and thus, the dynamic base fee is low), then spike suddenly for an NFT mint. This can be the case at a global level (as in Ethereum's TFM) and may be even more volatile at a local per-account level (as is considered in SIMD-0110).

However, Solana also benefits from its incredibly low block times here. These can allow the base fee to adjust more rapidly to a sudden demand shock, depending on how aggressively the curve is set to move. The shape of the fee controller here is incredibly important.

[The fact that this write-lock fee is charged on CUs requested

also properly incentivizes users and developers to accurately estimate a transaction's CU usage](https://github.com/solana-foundation/solana-improvement-documents/pull/110/files#r1460668318). This avoids the issue we discussed earlier where the current flat signature base has no penalty for requesting far more CUs than required (even up to the max of 1.4mm CUs). Otherwise, only the priority fee carries this incentive today (as it is also charged by CUs requested).

One potential criticism here is that account-based local-fee markets (especially this proposal, which requires an ongoing EMA to be calculated for every account) could be computationally expensive. This type of multidimensional fee is unbounded, given that any account can be congested, which would likely present difficulties for such a TFM. However, in SIMD-0110's case, this is avoided by setting an upper limit on the number of accounts whose CU usage EMA can be tracked at a given time.

**Efficiently Computable

: The block auction mechanism must be designed such that it can be efficiently computed for a given block producer (or builder) — Eclipse and Solana's slots are less than 400ms, which puts a tight constraint on maximum compute time for a given block.

Given that Solana block inclusion would still be non-deterministic even with this proposal implemented, there can still potentially be issues with users accurately updating their bids in real-time to ensure their transactions are included in blocks. Further addressing this requires changes to the scheduler as we discuss in the next section.

Changes to Solana's Default Scheduler

As discussed earlier, the "scheduler" is the default algorithm included in theSolana Labs validator client, which schedules transactions for execution and builds blocks continuously. It plays an incredibly important role in Solana's fee market even though its default behavior is not enforced in-protocol as validators may choose to run other algorithms. We'll focus here on the current scheduler and upcoming proposed changes, which are being worked on by Andrew Fitzgerald.

Solana's current scheduler introduces "jitter" in its handling of users' transactions by randomly assigning them to one of the four non-vote transaction threads (an additional two threads are reserved for processing vote transactions) before trying to sort the outstanding transactions by priority fee and check the relevant locks ('lock grab'), as the diagram below shows. Multiple batches of transactions are pulled to be allocated to threads during the "Banking Stage" — the process run by a Solana validator wherein transactions are processed and under which the scheduling process occurs.

Source: Andrew Fitzgerald, Solana Banking Stage and Scheduler

One significant issue with the default scheduler has been that during periods of intense network activity, the queue of each thread is often filled with conflicting transactions (say before an NFT mint or a widely anticipated token generation event). Each thread may contain transactions with the same or overlapping read- or write-locks, meaning those transactions must be rescheduled for later execution. The result of this, however, is that in the extreme worst case, only one of the four default scheduler threads could be executing transactions at a given time.

The crux of the upgrade to Solana's default scheduler lies in the transition from the legacy approach (named the ThreadLocalMultiIterator

mode) to the new approach to scheduling, named the CentralScheduler

mode. This article will only provide an overview and analysis of the changes. However, further information can be found in Andrew Fitzgerald's article and an accompanying summary blogpost by Harsh Patel from the Tiny Dancer team. An overview of the new scheduling process is shown below.

Source: Andrew Fitzgerald, Solana Banking Stage and Scheduler

The new scheduler involves a central single scheduler receiving transactions from the channel before checking for the relevant locks. After this point, transactions are assigned to particular parallel worker threads for execution. The central scheduler has a view of the various read- and write-locks used by a given worker thread, allowing it to determine the best thread for a new transaction. As transactions are executed and processed by a given worker thread, a message is sent back

to the central scheduler so it can reevaluate which parts of Solana's state are considered locked.

The scheduler uses an algorithm referred to as a "prio-graph," which is a direct acrylic graph that begins with the highest priority (fee) transactions and lines (or, more accurately, edges) between a given highest priority transactions and the following highest priority transactions which conflict with it due to overlapping locks. This is (tentatively) done for a "look-ahead" window of a preconfigured size of 2,048 (subject to change) transactions, which can be added to the graph — the following charts show the prio-graph works for a given set of transactions where edges between them represent conflicting locks.

In addition to adopting the prio-graph scheduler, this version introduces additional efficiencies to help reduce processing overhead, such as removing the redundant elements of the banking stage. The new scheduler should improve by significantly reducing the probability of failed writes and read-locking during periods of high activity on Solana. We could expect a reduction in jitter due to the new default scheduler. Still, given the continuous nature of the block-building process, there will continue to be non-determinism in block inclusion.

Program Rebatable Account Write (PRAW) Fees

Authored by Godmode Galactus and Max Schneider, SIMD-0016 proposes Program Rebatable Account Write (PRAW) fees. They would give significant control to application developers, as they could set the criteria for payment and rebates of these fees, allowing them to incentivize and disincentivize user behavior as they see fit.

Currently, Solana programs have no ability to penalize transactions for acquiring write locks over their state. PRAW fees would allow Solana account owners to charge for failed transactions that write-lock their state. These fees would be transferred to the writable account they are locking. However, account owners can set these fees such that they are then rebated

back to the user at the end of the transaction if it meets the specified criteria.

In particular, this can deter users from write-locking accounts that they do not actually use in the transaction's execution. This is currently possible given that Solana currently has no checks in place to see whether a given account would be used, a priori, by a given transaction that has write-locked it. PRAWs offer programs a way to disincentivize transactions that lock the program's state in an attempt to identify an opportunity with the intention to revert if the opportunity is no longer valid at the time of execution. These fees would be applied even if the transaction were to fail during execution.

Conversely, users can specify the maximum amount of PRAW fees they are willing to pay in a transaction. Any fee specified in the transaction above that of the current PRAW fee for a given write-locked account would be refunded.

Members of the Solana community flagged issues with this proposal: the ability for different programs to operate entirely autonomously seems suboptimal, and the ability to estimate fees accurately would prove difficult. Moreover, there are potentially simpler and more uniform ways to deal with these griefing issues around write-locked accounts, such as SIMD-0110.

**Griefing Resistance

: A subset of DSIC wherein users are not incentivized to misrepresent their access lists — misstating the resources required for their transaction [Gar23].

The PRAW proposal would potentially fail to prevent spam as it relies on application developers sufficiently: 1) being able to distinguish spam from "normal behavior" and 2) voluntarily choosing to charge more for a negative externality they are partially responsible for when it may not be in their best interest to do so, and they can simply choose not to.

In contrast, while members of the Solana research community are undeniably split on the introduction of EMA base fees, there is generally agreement around adding some component of the base fee which scales with CUs. This can incentivize accurate CU estimation and efficient usage of CUs by developers.

Parting Thoughts

Solana's unique engineering and performance goals require unique TFM considerations. Simply porting Ethereum's existing fee market to Solana is, of course, not the solution here, but there are valuable lessons to be learned from it. This is highly relevant for mechanisms that are both:

- In-protocol
- Consensus-enforced TFMs (e.g., EIP-1559 and SIMD-0110)
- Out-of-protocol
- PBS via MEV-Boost, Solana scheduler improvements, and Jito auctions

In-protocol

- Consensus-enforced TFMs (e.g., EIP-1559 and SIMD-0110)

Out-of-protocol

- PBS via MEV-Boost, Solana scheduler improvements, and Jito auctions

For both Solana and Ethereum, in-protocol and out-of-protocol mechanisms appear likely to co-exist and evolve together going forward. The balance between them is one of the fundamental questions in designing these systems. The debate surrounding SIMD-0110 often centers around two opposing views:

- Scheduler and networking improvements to reduce jitter will sufficiently address the problems described here, so major changes to the in-protocol TFM are unnecessary.
- While out-of-protocol scheduler and networking improvements are necessary, they areinherently insufficient. In-protocol economic backpressure is required.

Scheduler and networking improvements to reduce jitter will sufficiently address the problems described here, so major changes to the in-protocol TFM are unnecessary.

While out-of-protocol scheduler and networking improvements are necessary, they areinherently insufficient. In-protocol economic backpressure is required.

Multidimensional resource pricing in some form is clearly valuable in both cases as well. Ethereum is starting to pursue such a TFM at the base resource level, with EIP-4844 splitting out blob data from the execution market. Conversely, Solana is pushing ahead with multidimensional resource pricing at the individual account level to pioneer “local fee markets”.

The TFM research here is cutting-edge, and researchers are constantly finding new and innovative ways to improve how fees work on Solana and other chains. We’re optimistic that the proposals discussed here will all continue to make Solana even more efficient, scalable, user-friendly, and economically sustainable.

As Eclipse approaches our mainnet launch, we're also excited to share more on how we’ll apply this existing work to our own TFM, which will certainly continue to evolve for years to come. We intend to experiment with and push forward mechanisms in this area. An essential benefit of the modular paradigm is that it allows for easier cross-pollination of research and engineering from different ecosystems. This rate of experimentation will now only continue to increase, benefiting everyone building here for the long term.

Please follow the instructions here to get involved with Eclipse Testnet. If you have specific inquiries about our Testnet or any other technical questions, you can contact us on Twitter or Discord.